ФТФ, 2-ый курс, ИВТ-Б, Королёв Алексей.

**Ссылка на GitHub:**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Тема:** Наследование и полиморфизм. Абстрактные классы и интерфейсы. RTTI.

**Цель:** Научиться реализовывать на С++ наследование классов, программировать абстрактные классы и интерфейсы, виртуальные методы, а также динамически определять тип объекта во время выполнения программы.

**Индивидуальные задания:**

1. Написать в ООП-стиле код программы, позволяющей работать с арифметическими выражениями разного вида, оперирующими вещественными числами: вычислять результат выражения, выводить запись выражения на консоль и в файл лога. Например, для вычисления выражений вида (10+4+2+3+7+1) и (1+2.5) будет использоваться класс Summator, выражений вида (2\*3\*7\*1) – класс Multiplier, и т.д. В коде необходимо отразить следующее:

* Создать интерфейс ILoggable с 2 методами (функционал логирования): Запись лога всего выражения на консоль: void logToScreen() Добавление записи лога всего выражения в файл лога: void logToFile(const std::string& filename).
* Создать абстрактный класс ExpressionEvaluator, реализующий интерфейс ILoggable и предоставляющий чисто виртуальный метод double calculate() для вычисления результата произвольного выражения. Количество операндов должно храниться в отдельном члене класса. Сами операнды х1,х2,х3 и т.д. должны храниться в члене данного класса – массиве, в куче (динамической памяти).
* Класс ExpressionEvaluator должен предоставлять два конструктора и виртуальный деструктор. В конструкторе без параметров выделять память под 20 операндов и инициализировать их нулями, в конструкторе с параметром n – выделять память под n элементов и инициализировать нулями. Также необходимо реализовать 2 метода, позволяющие присвоить операндам конкретные значения: Присвоить значение value одному операнду на позиции pos: void setOperand(size\_t pos, double value) Заполнить сразу группу из n операндов массивом значений ops: void setOperands(double ops[], size\_t n)
* В деструкторе должна освобождаться память, выделенная в конструкторе.
* Создать два подкласса класса ExpressionEvaluator, работающих со стандартными выражениями, в соответствии с вариантом, из четырех возможных:

Summator – сумма всех операндов (х1 + х2 + х3 + х4 + ...)

Subtractor – разность всех операндов (х1 – х2 – х3 – х4 – ...)

Multiplier – произведение всех операндов (х1 \* х2 \* х3 \* х4 \* ...)

Divisor – частное всех операндов (х1/х2/х3/х4/...), но если хоть один операнд равен 0, то результату выражения присвоить также 0.

* Создать подкласс CustomExpressionEvaluator, работающий со специфическими выражениями, вид которых приведен в варианте.
* Подклассы ExpressionEvaluator, для которых порядок следования операндов важен, должны также реализовывать интерфейс IShuffle. Данный интерфейс объявляет 2 перегруженных метода (функционал перемешивания операндов): Произвольно перемешать операнды: void shuffle() Перемешать операнды, находящиеся на позициях i и j: void shuffle(size\_t i, size\_t j)

В функции main() необходимо продемонстрировать работу созданных классов:

* Создать массив из трех указателей на класс обработки арифметических выражений.
* В соответствии с вариантом, создать в куче три объекта конкретных подклассов обработки арифметических выражений и установить на них указатели; присвоить их операндам значения двумя способами (поэлементным и групповым).
* Продемонстрировать полиморфизм: организовать проход в цикле по указателям и вывести лог выражения на консоль и в файл (в консоли отобразить еще и сам результат выражения).
* Организовать цикл по указателям, в теле которого средствами С++ проверить, реализует ли текущий объект интерфейс IShuffle. Если да, то вызвать один из методов shuffle() этого объекта и отобразить на экране запись выражения после перемешивания операндов, а также вычислить и отобразить результат нового выражения.

*Вариант 5. Вид выражения CustomExpression: result = x1 + x2 \* x3 + x4 \* х5 + ... Порядок создания и инициализации объектов подклассов: Divisor: 4 операнда, присвоить поэлементно 150, -3, 10, –2.5. CustomExpressionEvaluator: 5 операндов, присвоить группой 5, 16, -3, 10, 12. Multiplier: 5 операндов, присвоить группой 1.5, 4, -2.5 -8, -15. Метод shuffle() – отсортировать все операнды в порядке возрастания. Метод shuffle(size\_t i, size\_t j) – если хотя бы один из i-ого и j-ого операндов имеет дробную часть, поменять их местами, иначе – не менять. Формат вывода:*

*5 + 16\*(-3) + 10\*12 < Total 5 >*

*< Result 77 >*

2. Дополнить код задания 3 лабораторной работы № 2, написав еще два класса по предметной области, в соответствии с вариантом. Продумать и корректно реализовать схему наследования классов. В главной функции продемонстрировать применение интерфейса, полиморфизм и RTTI на примере 3-4 объектов классов, по аналогии с заданием 1.

*Вариант 5. Класс КОНДИЦИОНЕР + классы БЫТОВОЕ УСТРОЙСТВО, ОБОГРЕВАТЕЛЬ. Реализовать схему наследования классов и корректно распределить по классам данные: фирма, модель, вес, температура, режим, год выпуска, мощность. Интерфейс возможности управления / регулировки устройства IControllable с методом void control(int temperature) – отрегулировать устройство в зависимости от установленной в параметре температуры. Реализация метода в классе кондиционера: если температура задана меньше 10 градусов, то выдать сообщение и выключиться, иначе присвоить текущему режиму разный номер в зависимости от температуры (т.е. выставить режим). Реализация метода в классе обогревателя: если задана температура выше 45 градусов, то выдать сообщение и выключиться, иначе присвоить текущему режиму разный номер в зависимости от температуры. В main() создать 2 кондиционера и 1 обогреватель, продемонстрировать полиморфизм control().*

**Ход работы:**

1. Код программы:
   1. Код CustomExpressionEvaluator.h:

#ifndef CUSTOM\_EXPRESSION\_EVALUATOR\_H

#define CUSTOM\_EXPRESSION\_EVALUATOR\_H

#include "ExpressionEvaluator.h"

#include <sstream>

// Класс для вычисления выражения вида x1 + x2\*x3 + x4\*x5 + ...

class CustomExpressionEvaluator : public ExpressionEvaluator {

public:

explicit CustomExpressionEvaluator(size\_t n) : ExpressionEvaluator(n) {}

// Вычисление результата пользовательского выражения

double calculate() override {

double result = operands[0];

for (size\_t i = 1; i < operandsCount - 1; i += 2) {

result += operands[i] \* operands[i + 1];

}

if (operandsCount % 2 == 0) {

result += operands[operandsCount - 1];

}

return result;

}

protected:

// Формирование строки выражения

std::string getExpressionString() const override {

std::ostringstream oss;

oss << operands[0];

for (size\_t i = 1; i < operandsCount - 1; i += 2) {

oss << " + " << operands[i] << "\*" << operands[i + 1];

}

if (operandsCount % 2 == 0) {

oss << " + " << operands[operandsCount - 1];

}

return oss.str();

}

};

#endif

* 1. Код Divisor.h:

#ifndef DIVISOR\_H

#define DIVISOR\_H

#include "ExpressionEvaluator.h"

#include "IShuffle.h"

#include <sstream>

#include <algorithm>

#include <cmath>

// Класс для вычисления частного всех операндов

class Divisor : public ExpressionEvaluator, public IShuffle {

public:

explicit Divisor(size\_t n) : ExpressionEvaluator(n) {}

// Вычисление результата деления

double calculate() override {

if (operandsCount == 0) return 0;

// Проверка деления на ноль

for (size\_t i = 0; i < operandsCount; ++i) {

if (i > 0 && operands[i] == 0) return 0;

}

double result = operands[0];

for (size\_t i = 1; i < operandsCount; ++i) {

result /= operands[i];

}

return result;

}

// Сортировка операндов по возрастанию

void shuffle() override {

std::sort(operands, operands + operandsCount);

}

// Перемещение операндов если есть дробная часть

void shuffle(size\_t i, size\_t j) override {

if (i >= operandsCount || j >= operandsCount) return;

double fractI = operands[i] - std::floor(operands[i]);

double fractJ = operands[j] - std::floor(operands[j]);

if (fractI > 0 || fractJ > 0) {

std::swap(operands[i], operands[j]);

}

}

protected:

// Формирование строки выражения

std::string getExpressionString() const override {

std::ostringstream oss;

for (size\_t i = 0; i < operandsCount; ++i) {

if (i > 0) oss << " / ";

oss << operands[i];

}

return oss.str();

}

};

#endif

* 1. Код ExpressionEvaluator.h:

#ifndef EXPRESSION\_EVALUATOR\_H

#define EXPRESSION\_EVALUATOR\_H

#include "ILoggable.h"

#include <cstddef>

#include <string>

#include <fstream>

#include <iostream>

// Абстрактный базовый класс для вычисления выражений

class ExpressionEvaluator : public ILoggable {

protected:

double\* operands; // Массив операндов

size\_t operandsCount; // Количество операндов

public:

// Конструктор по умолчанию - выделяет память под 20 операндов

ExpressionEvaluator() : operandsCount(20) {

operands = new double[operandsCount]();

}

// Конструктор с параметром - выделяет память под n операндов

explicit ExpressionEvaluator(size\_t n) : operandsCount(n) {

operands = new double[operandsCount]();

}

// Виртуальный деструктор для освобождения памяти

virtual ~ExpressionEvaluator() {

delete[] operands;

}

// Чисто виртуальная функция для вычисления результата выражения

virtual double calculate() = 0;

// Установка значения одного операнда

void setOperand(size\_t pos, double value) {

if (pos < operandsCount) {

operands[pos] = value;

}

}

// Установка значений группы операндов

void setOperands(double ops[], size\_t n) {

size\_t count = std::min(n, operandsCount);

for (size\_t i = 0; i < count; ++i) {

operands[i] = ops[i];

}

}

// Запись выражения в файл

void logToFile(const std::string& filename) override {

std::ofstream file(filename, std::ios::app);

if (file.is\_open()) {

file << getExpressionString() << " < Total " << operandsCount << " > " << std::endl;

file << "< Result " << calculate() << " >" << std::endl;

file.close();

}

}

// Вывод выражения на экран

void logToScreen() override {

std::cout << getExpressionString() << " < Total " << operandsCount << " > " << std::endl;

std::cout << "< Result " << calculate() << " >" << std::endl;

}

protected:

// Виртуальная функция для получения строкового представления выражения

virtual std::string getExpressionString() const = 0;

};

#endif

* 1. Код ILoggable.h:

#ifndef ILOGGABLE\_H

#define ILOGGABLE\_H

#include <string>

// Интерфейс для реализации функционала логирования

class ILoggable {

public:

// Вывод информации на экран

virtual void logToScreen() = 0;

// Запись информации в файл

virtual void logToFile(const std::string& filename) = 0;

virtual ~ILoggable() = default;

};

#endif

* 1. Код IShuffle.h:

#ifndef ISHUFFLE\_H

#define ISHUFFLE\_H

#include <cstddef>

// Интерфейс для реализации функционала перемешивания операндов

class IShuffle {

public:

// Перемешивание всех операндов

virtual void shuffle() = 0;

// Перемешивание двух конкретных операндов

virtual void shuffle(size\_t i, size\_t j) = 0;

virtual ~IShuffle() = default;

};

#endif

* 1. Код Multiplier.h:

#ifndef MULTIPLIER\_H

#define MULTIPLIER\_H

#include "ExpressionEvaluator.h"

#include "IShuffle.h"

#include <sstream>

#include <algorithm>

#include <cmath>

// Класс для вычисления произведения всех операндов

class Multiplier : public ExpressionEvaluator, public IShuffle {

public:

explicit Multiplier(size\_t n) : ExpressionEvaluator(n) {}

// Вычисление результата умножения

double calculate() override {

double result = 1.0;

for (size\_t i = 0; i < operandsCount; ++i) {

result \*= operands[i];

}

return result;

}

// Сортировка операндов по возрастанию

void shuffle() override {

std::sort(operands, operands + operandsCount);

}

// Перемещение операндов если есть дробная часть

void shuffle(size\_t i, size\_t j) override {

if (i >= operandsCount || j >= operandsCount) return;

double fractI = operands[i] - std::floor(operands[i]);

double fractJ = operands[j] - std::floor(operands[j]);

if (fractI > 0 || fractJ > 0) {

std::swap(operands[i], operands[j]);

}

}

protected:

// Формирование строки выражения

std::string getExpressionString() const override {

std::ostringstream oss;

for (size\_t i = 0; i < operandsCount; ++i) {

if (i > 0) oss << " \* ";

oss << operands[i];

}

return oss.str();

}

};

#endif

* 1. Код ConsoleApplication1.cpp:

#include "Divisor.h"

#include "CustomExpressionEvaluator.h"

#include "Multiplier.h"

#include <iostream>

#include <typeinfo>

int main() {

setlocale(LC\_ALL, "Russian");

// Создаем массив указателей на базовый класс

ExpressionEvaluator\* expressions[3];

// Создаем объекты согласно варианту

expressions[0] = new Divisor(4);

expressions[1] = new CustomExpressionEvaluator(5);

expressions[2] = new Multiplier(5);

// Инициализация Divisor поэлементно

expressions[0]->setOperand(0, 150);

expressions[0]->setOperand(1, -3);

expressions[0]->setOperand(2, 10);

expressions[0]->setOperand(3, -2.5);

// Инициализация CustomExpressionEvaluator группой значений

double customValues[] = {5, 16, -3, 10, 12};

expressions[1]->setOperands(customValues, 5);

// Инициализация Multiplier группой значений

double multiplierValues[] = {1.5, 4, -2.5, -8, -15};

expressions[2]->setOperands(multiplierValues, 5);

// Демонстрация полиморфизма

std::cout << "Исходные выражения и их результаты:" << std::endl;

for (int i = 0; i < 3; ++i) {

expressions[i]->logToScreen();

expressions[i]->logToFile("expressions.log");

std::cout << std::endl;

}

// Демонстрация работы с интерфейсом IShuffle

std::cout << "\nПроверка функционала перемешивания:" << std::endl;

for (int i = 0; i < 3; ++i) {

IShuffle\* shuffleable = dynamic\_cast<IShuffle\*>(expressions[i]);

if (shuffleable) {

std::cout << "До перемешивания: " << std::endl;

expressions[i]->logToScreen();

shuffleable->shuffle();

std::cout << "После перемешивания: " << std::endl;

expressions[i]->logToScreen();

shuffleable->shuffle(0, 1);

std::cout << "После перемешивания(0,1): " << std::endl;

expressions[i]->logToScreen();

std::cout << std::endl;

}

}

// Освобождение памяти

for (int i = 0; i < 3; ++i) {

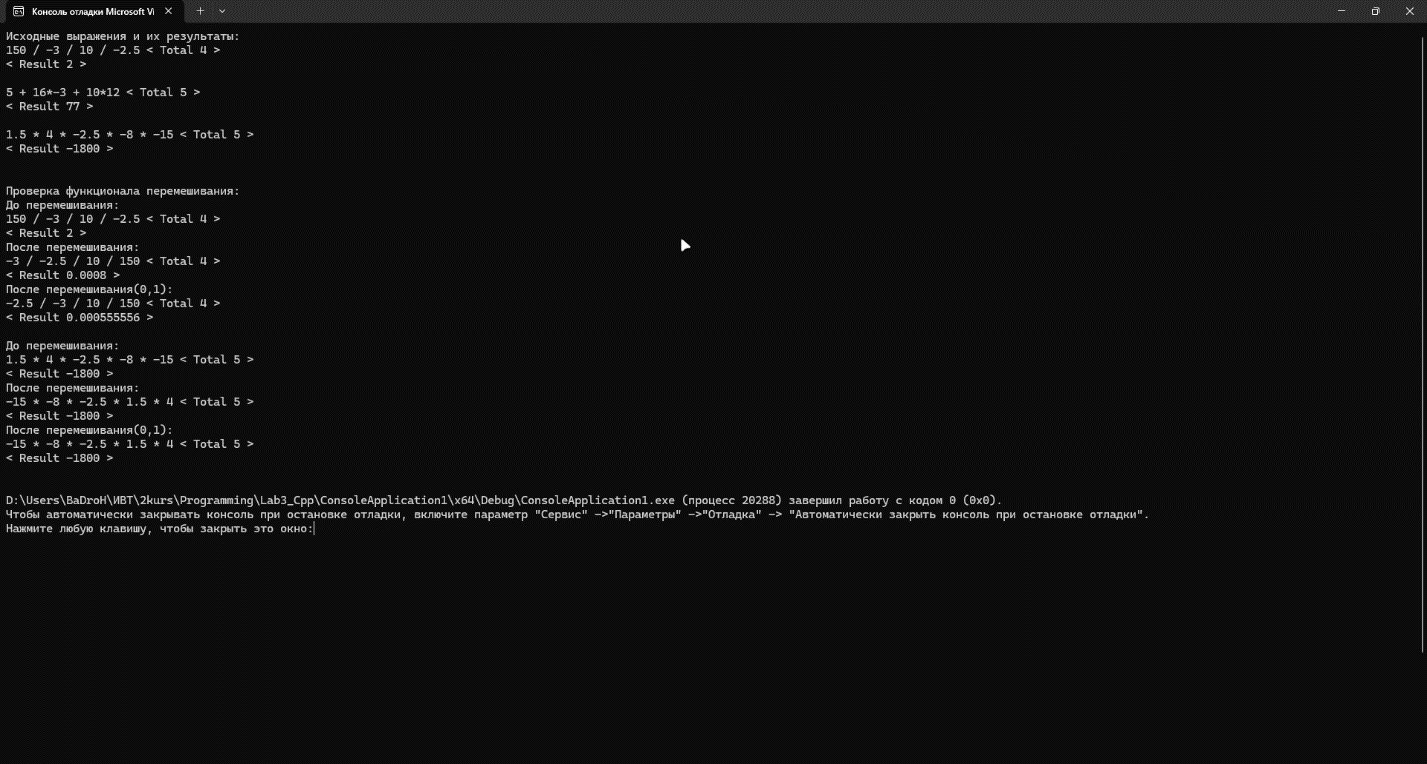
delete expressions[i];

}

return 0;

}

Результат программы:



1. Код программы:
   1. Код AirConditioner.h:

#ifndef AIR\_CONDITIONER\_H

#define AIR\_CONDITIONER\_H

#include "HouseholdDevice.h"

#include <string>

class AirConditioner : public HouseholdDevice {

private:

double temperature;

double totalTempChange;

int settingsCount;

public:

AirConditioner();

AirConditioner(const std::string& brand, const std::string& model,

double weight, int year, int power);

void control(int temperature) override;

void setTemperature(double temperature);

double getTemperature() const { return temperature; }

double getAverageTemperatureChange() const;

void serialize() const;

void deserialize();

void serialize(const std::string& filename) const;

void deserialize(const std::string& filename);

void printInfo() const override;

};

#endif

* 1. Код AirConditioner.cpp:

#include "AirConditioner.h"

#include <iostream>

AirConditioner::AirConditioner()

: temperature(0), totalTempChange(0), settingsCount(0) {}

AirConditioner::AirConditioner(const std::string& brand, const std::string& model,

double weight, int year, int power)

: HouseholdDevice(brand, model, weight, year, power),

temperature(0), totalTempChange(0), settingsCount(0) {}

void AirConditioner::control(int temperature) {

if (temperature < 10) {

std::cout << "Температура слишком низкая. Кондиционер выключается." << std::endl;

turnOff();

currentMode = 0;

}

else {

turnOn();

if (temperature < 18) {

currentMode = 1; // Слабое охлаждение

}

else if (temperature < 24) {

currentMode = 2; // Среднее охлаждение

}

else {

currentMode = 3; // Сильное охлаждение

}

setTemperature(temperature);

std::cout << "Кондиционер установлен на режим " << currentMode <<

" с температурой " << temperature << "°C" << std::endl;

}

}

void AirConditioner::setTemperature(double temp) {

totalTempChange += std::abs(temp - temperature);

temperature = temp;

settingsCount++;

}

double AirConditioner::getAverageTemperatureChange() const {

return settingsCount > 0 ? totalTempChange / settingsCount : 0;

}

void AirConditioner::printInfo() const {

HouseholdDevice::printInfo();

std::cout << "Текущая температура: " << temperature << "°C" << std::endl;

std::cout << "Среднее изменение температуры: " << getAverageTemperatureChange() << "°C" << std::endl;

}

// Пустые реализации для сериализации

void AirConditioner::serialize() const {}

void AirConditioner::deserialize() {}

void AirConditioner::serialize(const std::string&) const {}

void AirConditioner::deserialize(const std::string&) {}

* 1. Код Heater.h:

#ifndef HEATER\_H

#define HEATER\_H

#include "HouseholdDevice.h"

#include <string>

class Heater : public HouseholdDevice {

private:

double temperature;

bool hasThermostat;

public:

// Конструкторы

Heater();

Heater(const std::string& brand, const std::string& model,

double weight, int year, int power, bool hasThermostat = true);

// Методы управления

void control(int temperature) override;

void setTemperature(double temperature);

double getTemperature() const { return temperature; }

bool getHasThermostat() const { return hasThermostat; }

// Вывод информации

void printInfo() const override;

};

#endif

* 1. Код Heater.cpp:

#include "Heater.h"

#include <iostream>

Heater::Heater() : temperature(0), hasThermostat(true) {}

Heater::Heater(const std::string& brand, const std::string& model,

double weight, int year, int power, bool hasThermostat)

: HouseholdDevice(brand, model, weight, year, power),

temperature(0), hasThermostat(hasThermostat) {}

void Heater::control(int temperature) {

if (temperature > 45) {

std::cout << "Температура слишком высокая. Обогреватель выключается для безопасности." << std::endl;

turnOff();

currentMode = 0;

}

else {

turnOn();

if (temperature < 20) {

currentMode = 3; // Сильный нагрев

std::cout << "Температура низкая. Включен режим сильного нагрева." << std::endl;

}

else if (temperature < 30) {

currentMode = 2; // Средний нагрев

std::cout << "Включен режим среднего нагрева." << std::endl;

}

else {

currentMode = 1; // Слабый нагрев

std::cout << "Температура близка к целевой. Включен режим слабого нагрева." << std::endl;

}

setTemperature(temperature);

std::cout << "Обогреватель установлен на режим " << currentMode <<

" с целевой температурой " << temperature << "°C" << std::endl;

}

}

void Heater::setTemperature(double temp) {

temperature = temp;

}

void Heater::printInfo() const {

HouseholdDevice::printInfo();

std::cout << "Текущая целевая температура: " << temperature << "°C" << std::endl;

std::cout << "Наличие термостата: " << (hasThermostat ? "да" : "нет") << std::endl;

std::cout << "Текущий режим работы: ";

switch (currentMode) {

case 0: std::cout << "выключен"; break;

case 1: std::cout << "слабый нагрев"; break;

case 2: std::cout << "средний нагрев"; break;

case 3: std::cout << "сильный нагрев"; break;

default: std::cout << "неизвестен";

}

std::cout << std::endl;

}

* 1. Код HouseholdDevice.h:

#ifndef HOUSEHOLD\_DEVICE\_H

#define HOUSEHOLD\_DEVICE\_H

#include <string>

class IControllable {

public:

virtual void control(int temperature) = 0;

virtual ~IControllable() = default;

};

class HouseholdDevice : public IControllable {

protected:

std::string brand;

std::string model;

double weight;

int year;

int power;

bool isOn;

int currentMode;

public:

HouseholdDevice();

HouseholdDevice(const std::string& brand, const std::string& model,

double weight, int year, int power);

virtual ~HouseholdDevice() = default;

std::string getBrand() const { return brand; }

std::string getModel() const { return model; }

double getWeight() const { return weight; }

int getYear() const { return year; }

int getPower() const { return power; }

bool getIsOn() const { return isOn; }

int getCurrentMode() const { return currentMode; }

void setBrand(const std::string& brand) { this->brand = brand; }

void setModel(const std::string& model) { this->model = model; }

void setWeight(double weight) { this->weight = weight; }

void setYear(int year) { this->year = year; }

void setPower(int power) { this->power = power; }

virtual void turnOn() { isOn = true; }

virtual void turnOff() { isOn = false; }

virtual void printInfo() const;

};

#endif

* 1. Код HouseholdDevice.cpp:

#include "HouseholdDevice.h"

#include <iostream>

HouseholdDevice::HouseholdDevice()

: brand(""), model(""), weight(0), year(0), power(0), isOn(false), currentMode(0) {}

HouseholdDevice::HouseholdDevice(const std::string& brand, const std::string& model,

double weight, int year, int power)

: brand(brand), model(model), weight(weight), year(year), power(power), isOn(false), currentMode(0) {}

void HouseholdDevice::printInfo() const {

std::cout << "Бренд: " << brand << std::endl;

std::cout << "Модель: " << model << std::endl;

std::cout << "Вес: " << weight << " кг" << std::endl;

std::cout << "Год выпуска: " << year << std::endl;

std::cout << "Мощность: " << power << " Вт" << std::endl;

std::cout << "Состояние: " << (isOn ? "включен" : "выключен") << std::endl;

std::cout << "Текущий режим: " << currentMode << std::endl;

}

* 1. Код ConsoleApplication2.cpp:

#include "AirConditioner.h"

#include "Heater.h"

#include <iostream>

#include <iomanip>

#include <vector>

void printLine() {

std::cout << "\n----------------------------------------\n";

}

void demonstrateDevice(const std::string& deviceName, HouseholdDevice& device,

std::vector<int> temperatures) {

printLine();

std::cout << "=== Демонстрация работы устройства: " << deviceName << " ===" << std::endl;

std::cout << "\nНачальное состояние устройства:" << std::endl;

device.printInfo();

for (int temp : temperatures) {

printLine();

std::cout << "Тест: Установка температуры " << temp << "°C" << std::endl;

device.control(temp);

device.printInfo();

}

}

void demonstrateClimateControl() {

// Создаем устройства

AirConditioner premiumAC("Samsung", "WindFree Premium", 12.5, 2024, 2200);

Heater smartHeater("Ballu", "Smart Pro", 5.2, 2024, 2500, true);

// Температуры для тестирования кондиционера

std::vector<int> acTemperatures = {

5, // Слишком холодно - должен выключиться

22, // Комфортная температура - среднее охлаждение

26, // Жарко - сильное охлаждение

20 // Возврат к комфортной температуре

};

// Температуры для тестирования обогревателя

std::vector<int> heaterTemperatures = {

50, // Слишком горячо - должен выключиться

18, // Прохладно - сильный нагрев

25, // Комфортная температура - средний нагрев

15 // Холодно - сильный нагрев

};

std::cout << "\n=== ТЕСТИРОВАНИЕ КЛИМАТИЧЕСКОГО ОБОРУДОВАНИЯ ===\n" << std::endl;

// Демонстрация кондиционера

demonstrateDevice("Кондиционер Samsung WindFree Premium", premiumAC, acTemperatures);

// Демонстрация обогревателя

demonstrateDevice("Обогреватель Ballu Smart Pro", smartHeater, heaterTemperatures);

// Итоговая статистика

printLine();

std::cout << "=== ИТОГОВАЯ СТАТИСТИКА ===\n" << std::endl;

if (auto\* ac = dynamic\_cast<AirConditioner\*>(&premiumAC)) {

std::cout << "Кондиционер - среднее изменение температуры: "

<< std::fixed << std::setprecision(2)

<< ac->getAverageTemperatureChange() << "°C" << std::endl;

}

std::cout << "\nСтатус устройств:" << std::endl;

std::cout << "Кондиционер: " << (premiumAC.getIsOn() ? "включен" : "выключен")

<< ", режим " << premiumAC.getCurrentMode() << std::endl;

std::cout << "Обогреватель: " << (smartHeater.getIsOn() ? "включен" : "выключен")

<< ", режим " << smartHeater.getCurrentMode() << std::endl;

}

int main() {

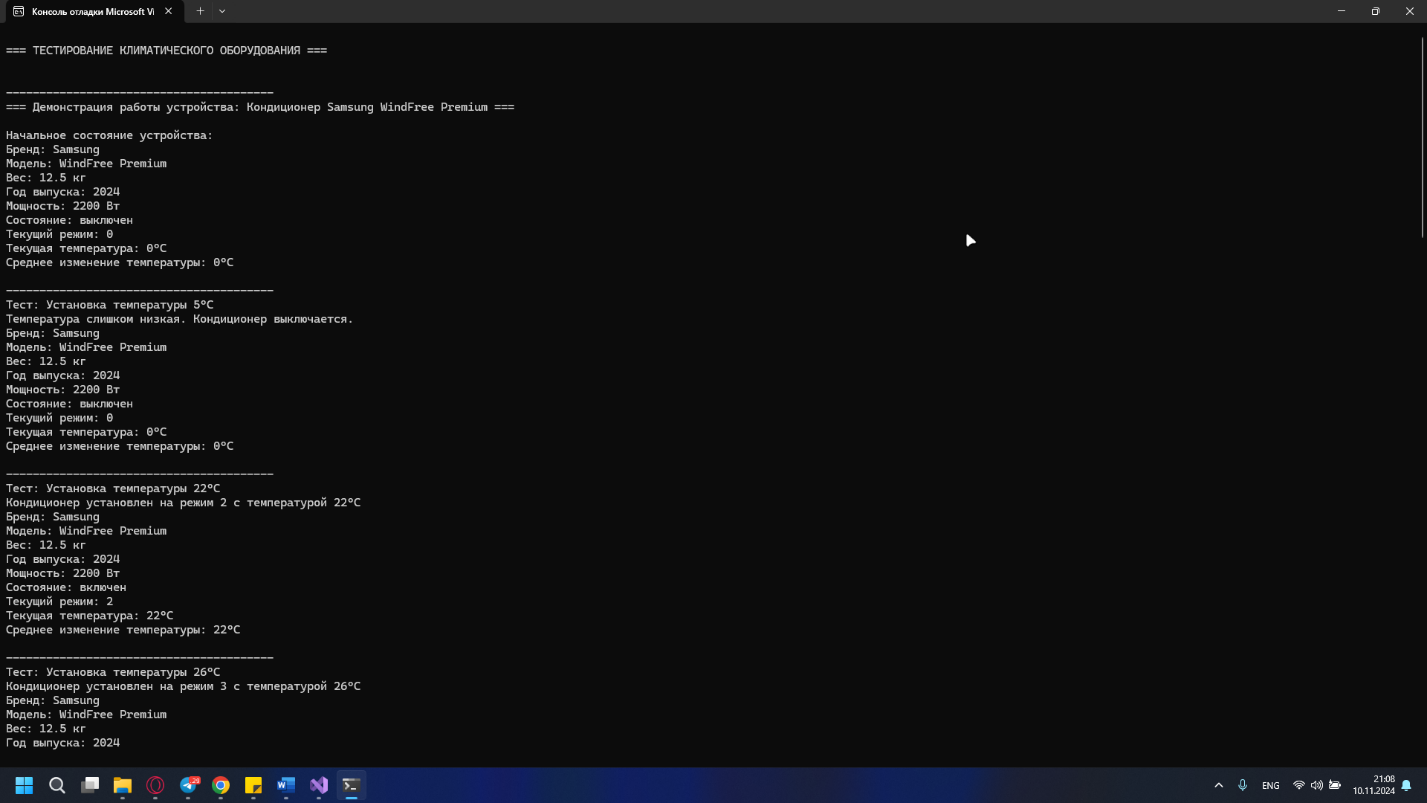
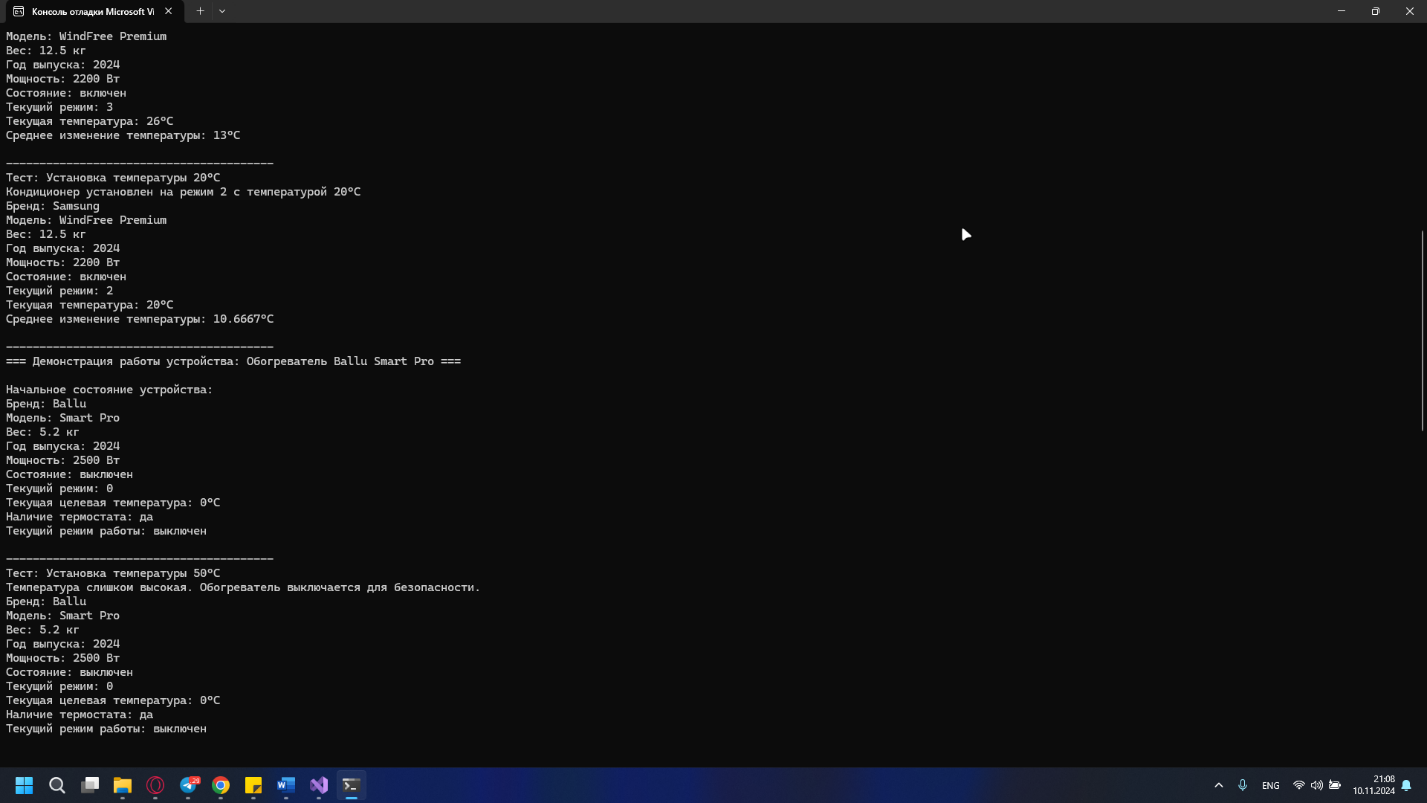
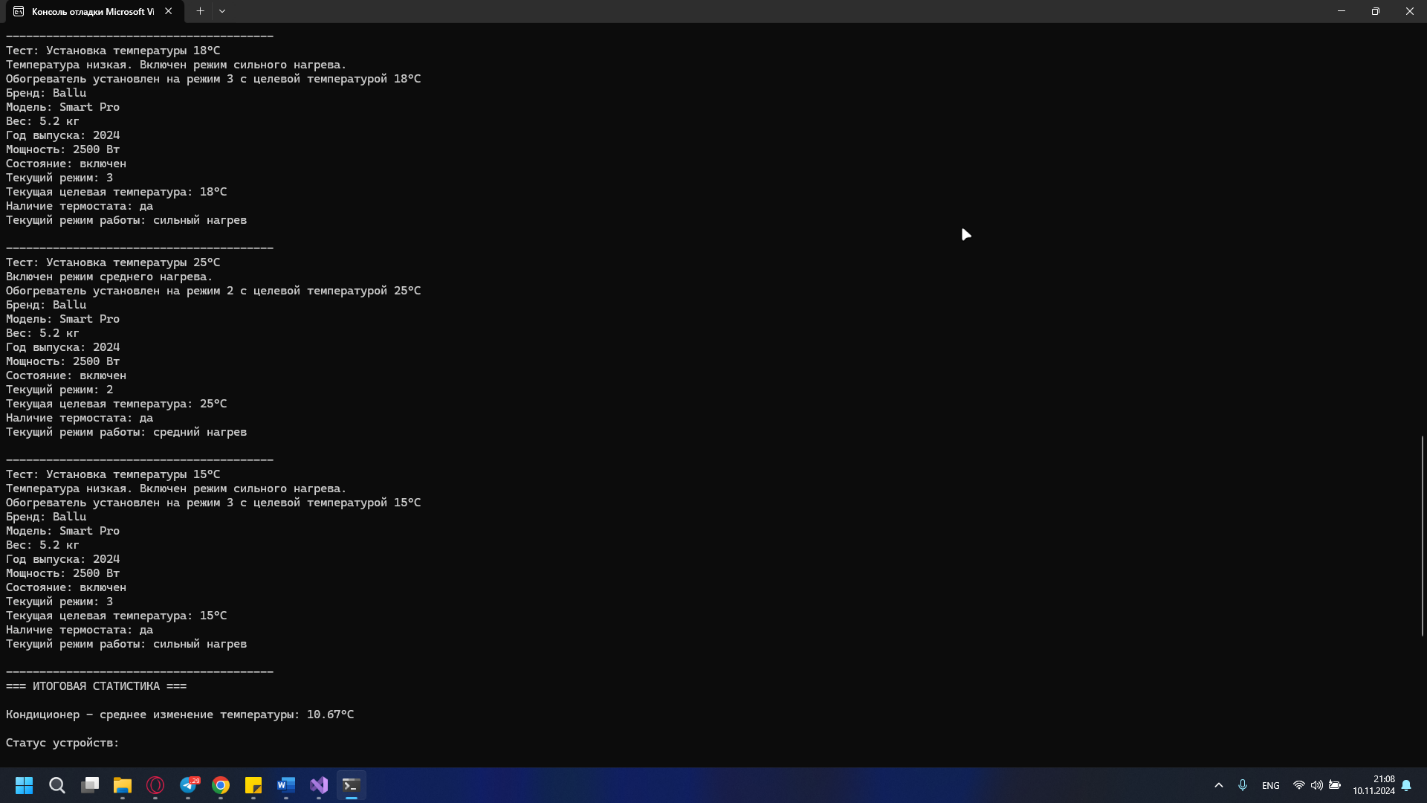
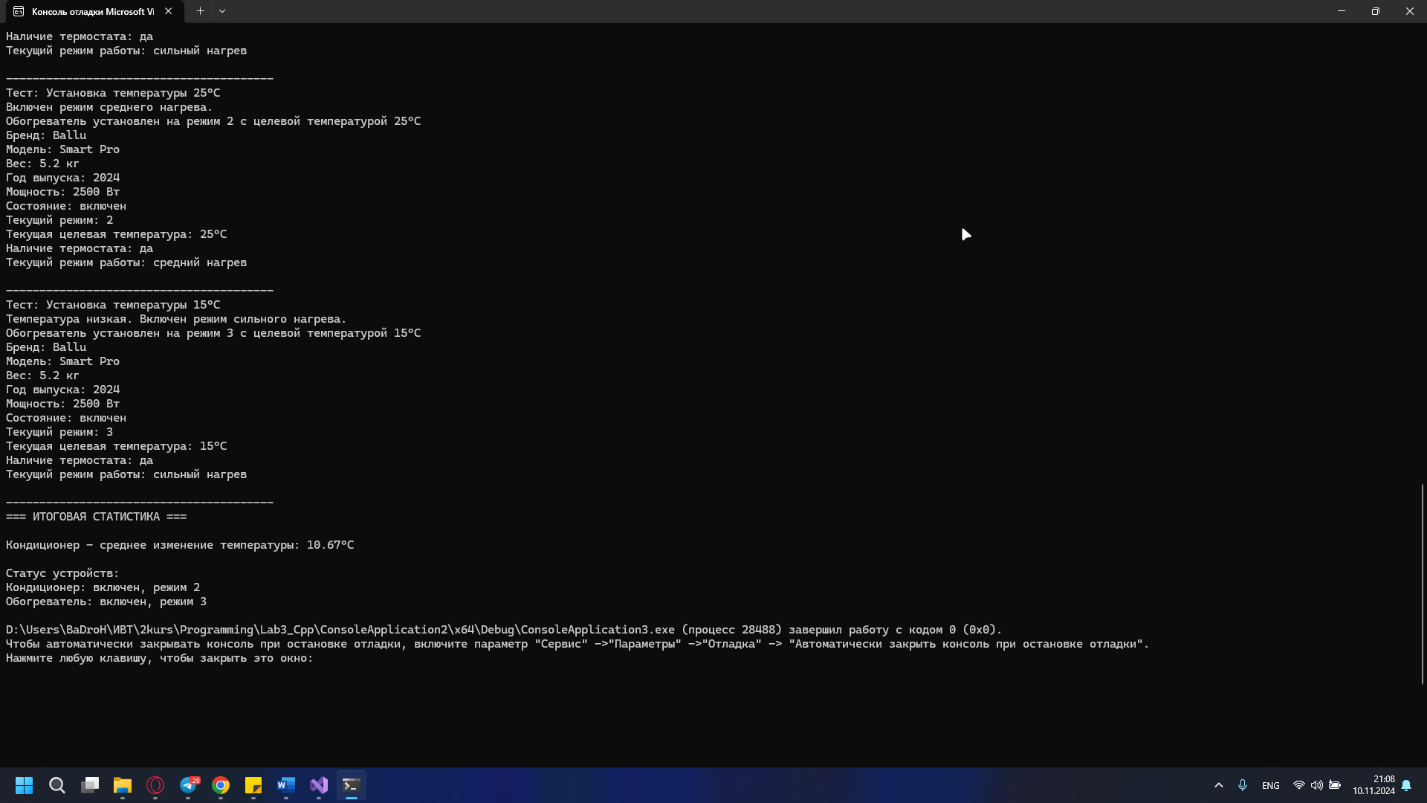
setlocale(LC\_ALL, "Russian");

demonstrateClimateControl();

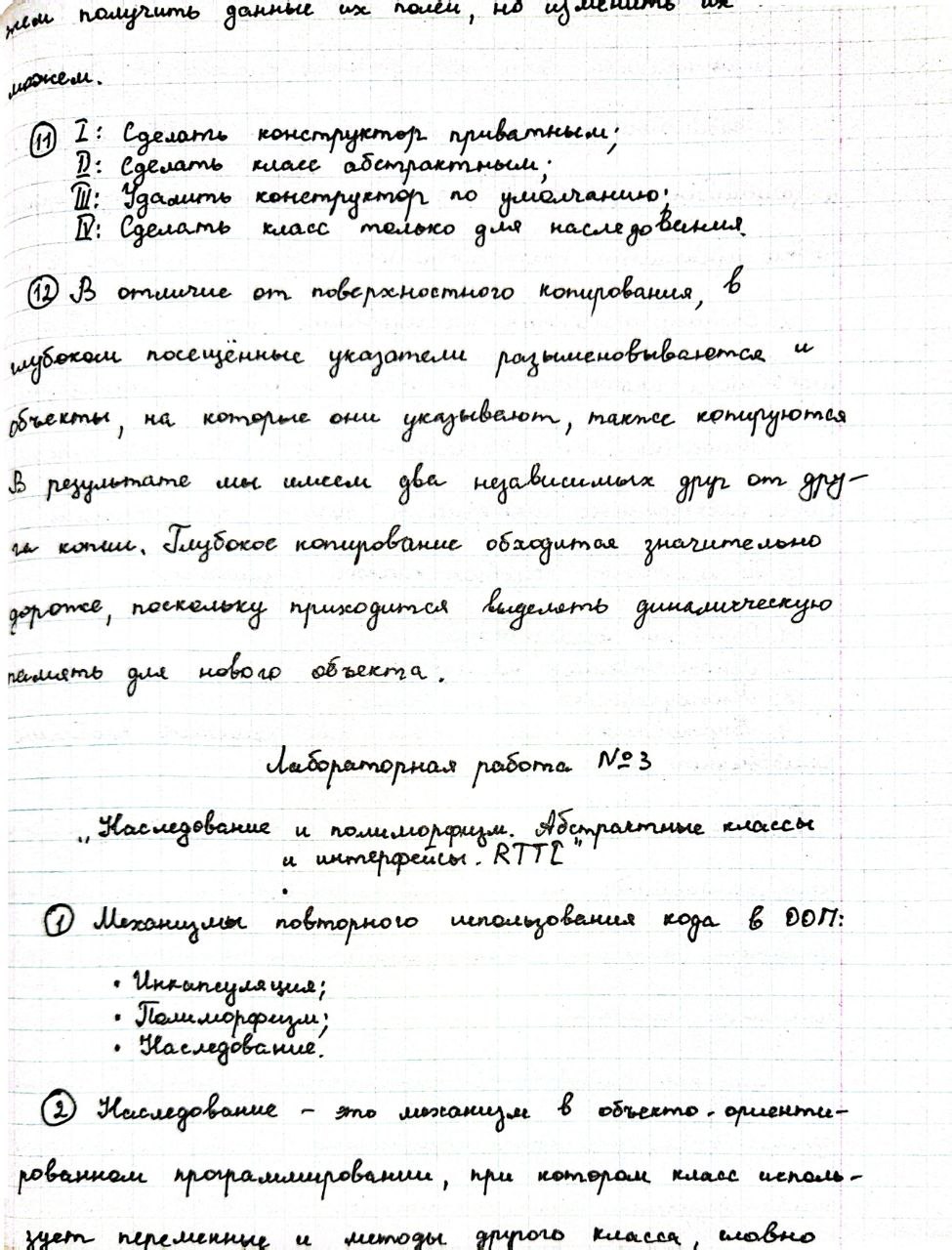
return 0;

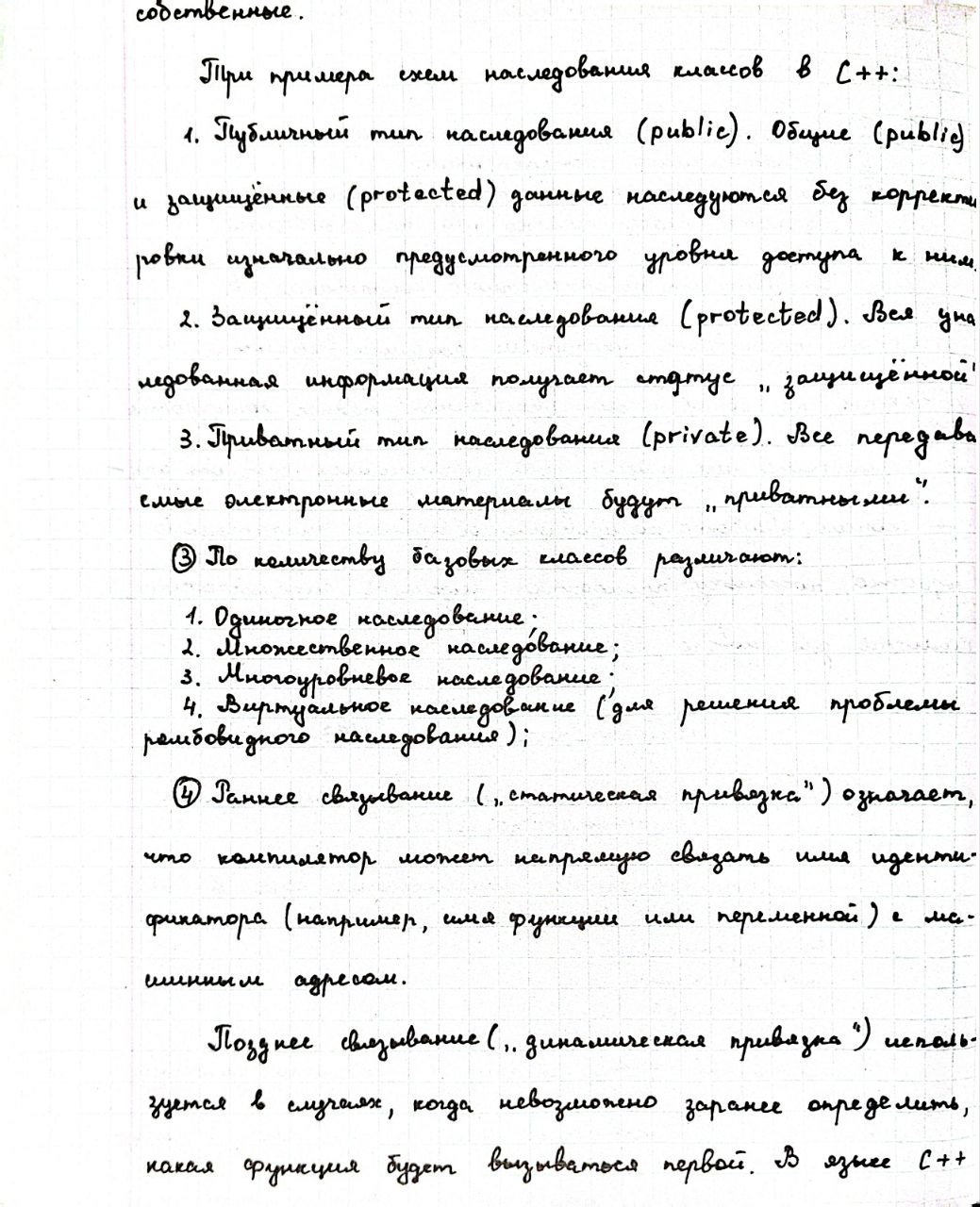
}

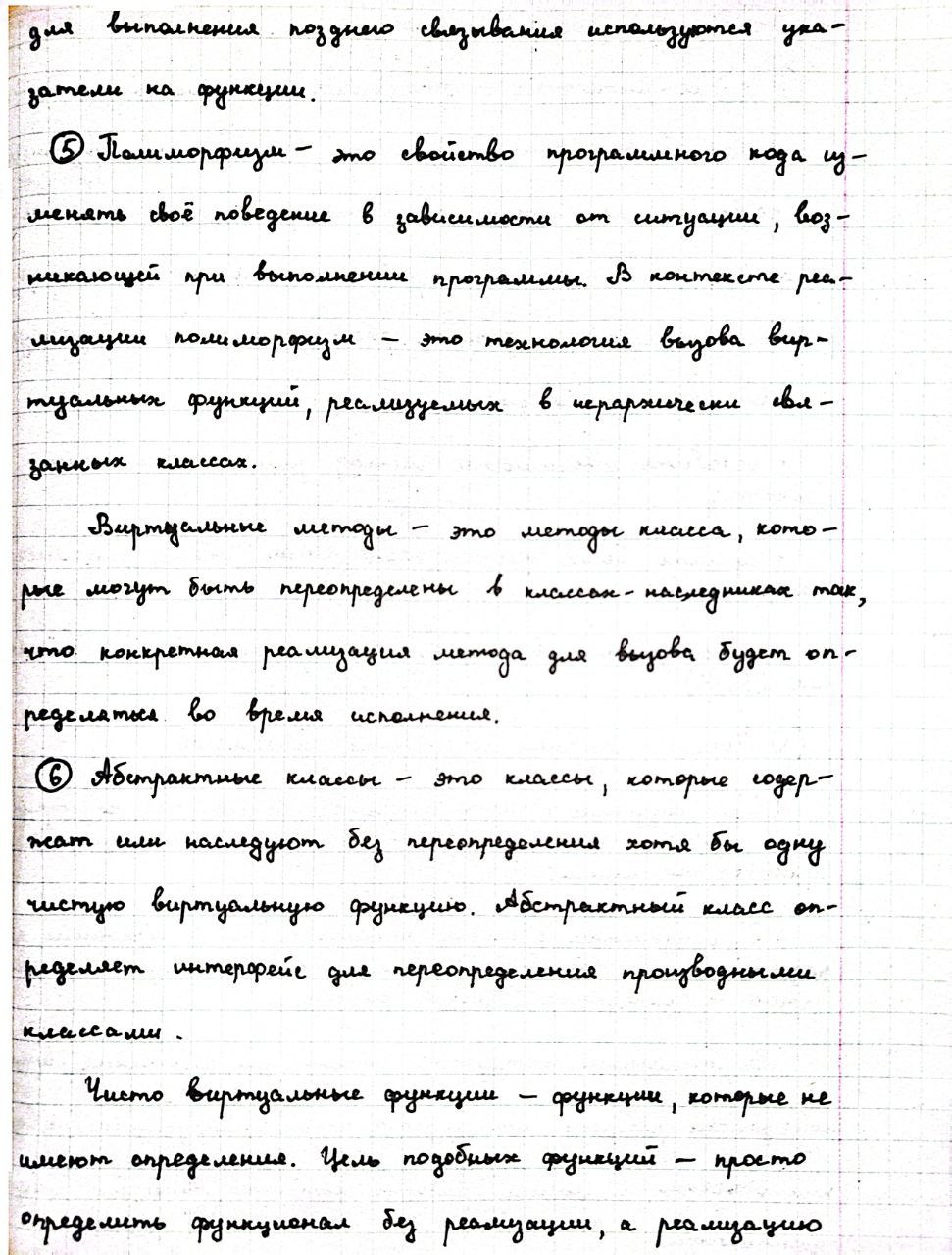
Результат программы:

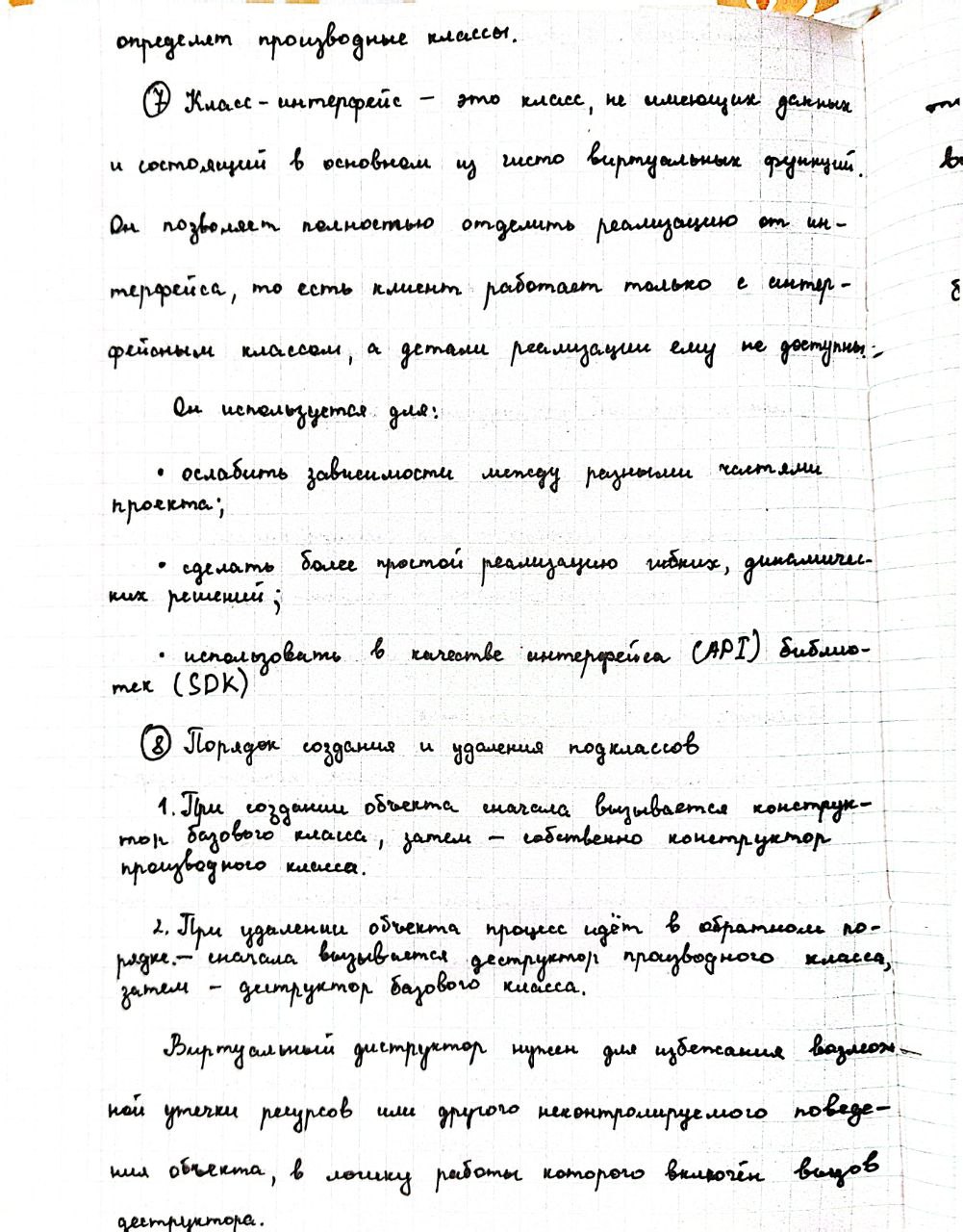


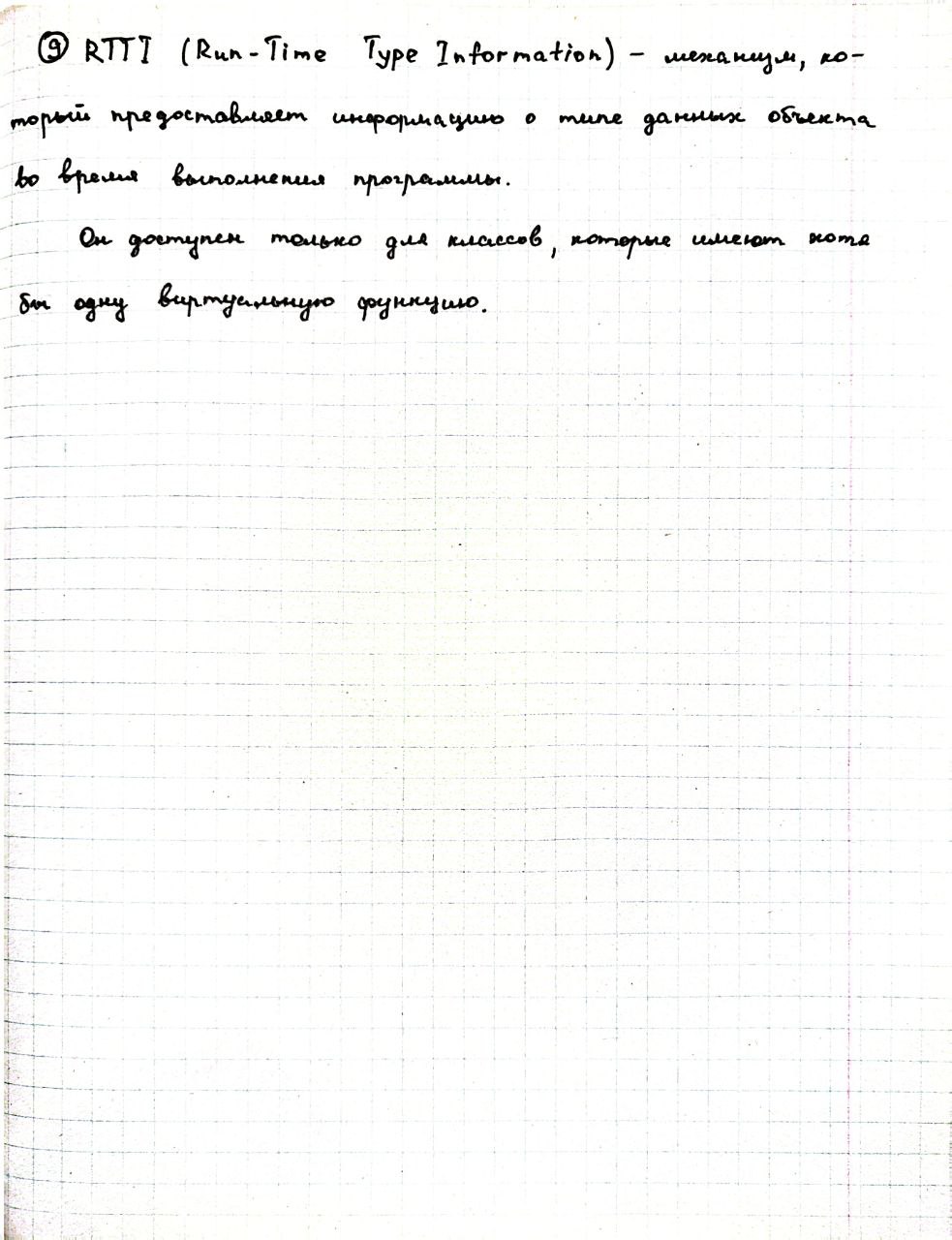
Контрольные вопросы:











Вывод: научился реализовывать на С++ наследование классов, программировать абстрактные классы и интерфейсы, виртуальные методы, а также динамически определять тип объекта во время выполнения программы.